autogluon_each_location

October 19, 2023

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*30
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = False

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
use_test_data = True
tune_and_test_length = 0.5 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
 ↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = True


shift_predictions_by_average_of_negatives_then_clip = False
```

```python
clip_predictions = True
shift_predictions = False
```

[2]:
```python
import pandas as pd
import numpy as np




import warnings
warnings.filterwarnings("ignore")



def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")



    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
       'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
       'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    X_shifted = X[X.index.minute==0][columns].copy()
    # loop through all rows and check if index + 1 hour is in the index, if so␣
↪get that value, else nan
    count1 = 0
    count2 = 0
    for i in range(len(X_shifted)):
        if X_shifted.index[i] + pd.Timedelta('1 hour') in X.index:
            count1 += 1
            X_shifted.iloc[i] = X.loc[X_shifted.index[i] + pd.Timedelta('1␣
↪hour')][columns]
        else:
            count2 += 1
            X_shifted.iloc[i] = np.nan

    print("COUNT1", count1)
    print("COUNT2", count2)

    X_old_unshifted = X[X.index.minute==0][columns]
    # rename X_old_unshifted columns to have _not_shifted at the end
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    # put the shifted columns back into the original dataframe
```

```python
    #X[columns] = X_shifted[columns]



    date_calc = None
    if "date_calc" in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
    # Convert 'date_forecast' to datetime format and replace original column␣
 ↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
```

```
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train, location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train = handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index - pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
        X_test["estimated_diff_hours"] = (X_test.index - pd.to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] = X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
```

```python
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
 ↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
 ↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

1 Feature enginering

[3]:
```
import numpy as np
import pandas as pd

X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)


for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,␣
↪multiply sample_weight with sample_weight_may_july if the ds is between␣
↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to␣
↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=␣
↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))


if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()
```

```
        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())




to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
↪gm3", "air_density_2m:kgm3", "msl_pressure:hPa", "pressure_100m:hPa",␣
↪"pressure_50m:hPa", "clear_sky_rad:W"]

X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.825
air_density_2m:kgm3                   1.245
ceiling_height_agl:m            2085.774902
clear_sky_energy_1h:J           1685498.875
clear_sky_rad:W                  452.100006
cloud_base_agl:m                2085.774902
dew_or_rime:idx                         0.0
dew_point_2m:K                   280.549988
diffuse_rad:W                    140.800003
diffuse_rad_1h:J                 538581.625
direct_rad:W                     102.599998
direct_rad_1h:J                 439453.8125
effective_cloud_cover:p           71.849998
elevation:m                             6.0
```

```
fresh_snow_12h:cm                      0.0
fresh_snow_1h:cm                       0.0
fresh_snow_24h:cm                      0.0
fresh_snow_3h:cm                       0.0
fresh_snow_6h:cm                       0.0
is_day:idx                             1.0
is_in_shadow:idx                       0.0
msl_pressure:hPa               1026.349976
precip_5min:mm                         0.0
precip_type_5min:idx                   0.0
pressure_100m:hPa              1013.325012
pressure_50m:hPa               1019.450012
prob_rime:p                            0.0
rain_water:kgm2                        0.0
relative_humidity_1000hPa:p      77.099998
sfc_pressure:hPa               1025.550049
snow_density:kgm3                      NaN
snow_depth:cm                          0.0
snow_drift:idx                         0.0
snow_melt_10min:mm                     0.0
snow_water:kgm2                        0.0
sun_azimuth:d                    93.415253
sun_elevation:d                  27.633499
super_cooled_liquid_water:kgm2       0.025
t_1000hPa:K                        282.625
total_cloud_cover:p              71.849998
visibility:m                     44177.875
wind_speed_10m:ms                    2.675
wind_speed_u_10m:ms                   -2.3
wind_speed_v_10m:ms                   -1.4
wind_speed_w_1000hPa:ms                0.0
y                                  2991.12
location                                 A
Name: 2019-06-11 06:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                       7.7              1.22825

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951                    0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00              0.0       1728.949951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
```

```
2019-06-02 22:00:00      280.299988            0.0               0.0  …

                     super_cooled_liquid_water:kgm2  t_1000hPa:K  \
ds
2019-06-02 22:00:00                             0.0   286.225006

                     total_cloud_cover:p  visibility:m  wind_speed_10m:ms  \
ds
2019-06-02 22:00:00                100.0  40386.476562                3.6

                     wind_speed_u_10m:ms  wind_speed_v_10m:ms  \
ds
2019-06-02 22:00:00               -3.575                 -0.5

                     wind_speed_w_1000hPa:ms    y  location
ds
2019-06-02 22:00:00                      0.0  0.0         A

[1 rows x 47 columns]
```

[5]:

```
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
#import matplotlib.pyplot as plt
#import seaborn as sns
#sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
#plt.show()
```

[6]:

```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
import numpy as np

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
```

```
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[7]:
```python
from autogluon.tabular import TabularDataset, TabularPredictor
from autogluon.timeseries import TimeSeriesDataFrame
import numpy as np
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
test_set = TabularDataset(data[data["ds"] >= split_time])

# shuffle test_set and only grab tune_and_test_length percent of it, rest goes␣
↪to train_set
test_set, new_train_set = split_and_shuffle_data(test_set, 40,␣
↪tune_and_test_length)


print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))
print("Length of test set", len(test_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = None
if use_tune_data:
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
```

```
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data =␣
↪split_and_shuffle_data(loc_test_set, 40, 0.5)
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)


else:
    if use_test_data:
        test_data = test_set
        print("Shape of test", test_data.shape[0])


train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Length of train set before adding test set 82026
Length of train set after adding test set 87486
Length of test set 5459
```

```
Shapes of tuning and test 2728 2731 5459
```

[8]:
```python
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
 ↪label="y", sample=None)
```

train_data dataset summary

```
                             count  unique                  top   freq  \
ceiling_height_agl:m         72280   59980
clear_sky_energy_1h:J        87486   46359
cloud_base_agl:m             81454   61360
diffuse_rad:W                87486   11092
diffuse_rad_1h:J             87486   46319
direct_rad:W                 87486   14181
direct_rad_1h:J              87486   40118
ds                           87486   36794  2021-02-05 14:00:00      3
effective_cloud_cover:p      87486    5655
elevation:m                  87486       3
fresh_snow_1h:cm             87486      39
fresh_snow_3h:cm             87486      70
fresh_snow_6h:cm             87486      96
is_day:idx                   87486       5
is_in_shadow:idx             87486       5
location                     87486       3                    A  31872
precip_type_5min:idx         87486      15
relative_humidity_1000hPa:p  87486    3799
sfc_pressure:hPa             87486    3795
snow_depth:cm                87486     487
snow_water:kgm2              87486     161
sun_azimuth:d                87486   83179
sun_elevation:d              87486   72262
super_cooled_liquid_water:kgm2  87486      53
t_1000hPa:K                  87486    1989
total_cloud_cover:p          87486    5556
visibility:m                 87486   85949
wind_speed_10m:ms            87486     596
y                            87486   11321

                                 first                 last           mean  \
ceiling_height_agl:m               NaT                  NaT    2861.929806
clear_sky_energy_1h:J              NaT                  NaT  530297.395771
cloud_base_agl:m                   NaT                  NaT    1740.241802
diffuse_rad:W                      NaT                  NaT      40.267497
diffuse_rad_1h:J                   NaT                  NaT    145328.6257
direct_rad:W                       NaT                  NaT      51.524847
direct_rad_1h:J                    NaT                  NaT   185338.05854
ds                          2019-01-01  2023-04-30 22:00:00
```

effective_cloud_cover:p	NaT	NaT	67.052836
elevation:m	NaT	NaT	11.414718
fresh_snow_1h:cm	NaT	NaT	0.008783
fresh_snow_3h:cm	NaT	NaT	0.026713
fresh_snow_6h:cm	NaT	NaT	0.05322
is_day:idx	NaT	NaT	0.490147
is_in_shadow:idx	NaT	NaT	0.556952
location	NaT	NaT	
precip_type_5min:idx	NaT	NaT	0.084976
relative_humidity_1000hPa:p	NaT	NaT	73.779918
sfc_pressure:hPa	NaT	NaT	1008.035963
snow_depth:cm	NaT	NaT	0.197574
snow_water:kgm2	NaT	NaT	0.090839
sun_azimuth:d	NaT	NaT	179.584247
sun_elevation:d	NaT	NaT	-0.705998
super_cooled_liquid_water:kgm2	NaT	NaT	0.058256
t_1000hPa:K	NaT	NaT	279.675551
total_cloud_cover:p	NaT	NaT	73.72398
visibility:m	NaT	NaT	32944.238197
wind_speed_10m:ms	NaT	NaT	3.032943
y	NaT	NaT	294.447861

	std	min	25%	50%	\
ceiling_height_agl:m	2532.377528	27.8	1082.3125	1856.075	
clear_sky_energy_1h:J	831839.646633	0.0	0.0	9084.9	
cloud_base_agl:m	1808.519208	27.5	598.15625	1174.775	
diffuse_rad:W	61.119566	0.0	0.0	1.35	
diffuse_rad_1h:J	218036.296903	0.0	0.0	12531.2	
direct_rad:W	114.236728	0.0	0.0	0.0	
direct_rad_1h:J	406379.39471	0.0	0.0	0.0	
ds					
effective_cloud_cover:p	34.132847	0.0	42.125	79.7	
elevation:m	7.881545	6.0	6.0	7.0	
fresh_snow_1h:cm	0.110515	0.0	0.0	0.0	
fresh_snow_3h:cm	0.277575	0.0	0.0	0.0	
fresh_snow_6h:cm	0.474579	0.0	0.0	0.0	
is_day:idx	0.486133	0.0	0.0	0.5	
is_in_shadow:idx	0.484138	0.0	0.0	1.0	
location					
precip_type_5min:idx	0.32995	0.0	0.0	0.0	
relative_humidity_1000hPa:p	14.200631	19.575	64.4	76.15	
sfc_pressure:hPa	13.038723	941.55	1000.05	1009.0	
snow_depth:cm	1.28439	0.0	0.0	0.0	
snow_water:kgm2	0.240248	0.0	0.0	0.0	
sun_azimuth:d	97.419022	6.983	94.4135	180.00663	
sun_elevation:d	24.006117	-49.932	-17.969875	-0.453875	
super_cooled_liquid_water:kgm2	0.106959	0.0	0.0	0.0	
t_1000hPa:K	6.551665	258.025	275.1	278.975	

```
total_cloud_cover:p                33.851299      0.0       53.475     92.825
visibility:m                     17949.64428  132.375   16564.5125   36910.75
wind_speed_10m:ms                   1.758713    0.025        1.675        2.7
y                                 774.531815     -0.0          0.0        0.0

                                     75%          max          dtypes  \
ceiling_height_agl:m              3916.78125  12285.775         float64
clear_sky_energy_1h:J             831169.675  3006697.2         float64
cloud_base_agl:m                  2080.39375  11673.725         float64
diffuse_rad:W                          67.15     334.75         float64
diffuse_rad_1h:J                  243365.275  1182265.4         float64
direct_rad:W                          32.225      683.4         float64
direct_rad_1h:J                   122454.125  2445897.0         float64
ds                                                       datetime64[ns]
effective_cloud_cover:p                 98.5      100.0         float64
elevation:m                             24.0       24.0         float64
fresh_snow_1h:cm                         0.0        7.1         float64
fresh_snow_3h:cm                         0.0       20.6         float64
fresh_snow_6h:cm                         0.0       34.0         float64
is_day:idx                               1.0        1.0         float64
is_in_shadow:idx                         1.0        1.0         float64
location                                                         object
precip_type_5min:idx                     0.0        5.0         float64
relative_humidity_1000hPa:p           85.175      100.0         float64
sfc_pressure:hPa                      1017.1   1043.725         float64
snow_depth:cm                            0.0       18.2         float64
snow_water:kgm2                          0.1       5.65         float64
sun_azimuth:d                     264.601138  348.48752         float64
sun_elevation:d                    16.004499   49.94375         float64
super_cooled_liquid_water:kgm2           0.1      1.375         float64
t_1000hPa:K                          284.225     303.25         float64
total_cloud_cover:p                     99.9      100.0         float64
visibility:m                        48289.05   75326.58         float64
wind_speed_10m:ms                       4.05     13.275         float64
y                                   183.7125    5733.42         float64

                               missing_count missing_ratio  raw_type  \
ceiling_height_agl:m                   15206      0.173811     float
clear_sky_energy_1h:J                                          float
cloud_base_agl:m                        6032      0.068948     float
diffuse_rad:W                                                  float
diffuse_rad_1h:J                                               float
direct_rad:W                                                   float
direct_rad_1h:J                                                float
ds                                                          datetime
effective_cloud_cover:p                                        float
elevation:m                                                    float
fresh_snow_1h:cm                                               float
```

```
fresh_snow_3h:cm                                                float
fresh_snow_6h:cm                                                float
is_day:idx                                                      float
is_in_shadow:idx                                                float
location                                                       object
precip_type_5min:idx                                            float
relative_humidity_1000hPa:p                                     float
sfc_pressure:hPa                                                float
snow_depth:cm                                                   float
snow_water:kgm2                                                 float
sun_azimuth:d                                                   float
sun_elevation:d                                                 float
super_cooled_liquid_water:kgm2                                  float
t_1000hPa:K                                                     float
total_cloud_cover:p                                             float
visibility:m                                                    float
wind_speed_10m:ms                                               float
y                                                               float

                               variable_type special_types
ceiling_height_agl:m                 numeric
clear_sky_energy_1h:J                numeric
cloud_base_agl:m                     numeric
diffuse_rad:W                        numeric
diffuse_rad_1h:J                     numeric
direct_rad:W                         numeric
direct_rad_1h:J                      numeric
ds
effective_cloud_cover:p              numeric
elevation:m                         category
fresh_snow_1h:cm                     numeric
fresh_snow_3h:cm                     numeric
fresh_snow_6h:cm                     numeric
is_day:idx                          category
is_in_shadow:idx                    category
location                            category
precip_type_5min:idx                category
relative_humidity_1000hPa:p          numeric
sfc_pressure:hPa                     numeric
snow_depth:cm                        numeric
snow_water:kgm2                      numeric
sun_azimuth:d                        numeric
sun_elevation:d                      numeric
super_cooled_liquid_water:kgm2       numeric
t_1000hPa:K                          numeric
total_cloud_cover:p                  numeric
visibility:m                         numeric
wind_speed_10m:ms                    numeric
```

y	numeric

test_data dataset summary

	count	unique	top	freq	\
ceiling_height_agl:m	2100	2065			
clear_sky_energy_1h:J	2731	1138			
cloud_base_agl:m	2374	2332			
diffuse_rad:W	2731	983			
diffuse_rad_1h:J	2731	1138			
direct_rad:W	2731	750			
direct_rad_1h:J	2731	932			
ds	2731	2200	2023-04-10 19:00:00	3	
effective_cloud_cover:p	2731	1348			
elevation:m	2731	3			
fresh_snow_1h:cm	2731	19			
fresh_snow_3h:cm	2731	36			
fresh_snow_6h:cm	2731	50			
is_day:idx	2731	5			
is_in_shadow:idx	2731	5			
location	2731	3	A	1094	
precip_type_5min:idx	2731	10			
relative_humidity_1000hPa:p	2731	1523			
sfc_pressure:hPa	2731	1575			
snow_depth:cm	2731	61			
snow_water:kgm2	2731	61			
sun_azimuth:d	2731	2716			
sun_elevation:d	2731	2652			
super_cooled_liquid_water:kgm2	2731	29			
t_1000hPa:K	2731	774			
total_cloud_cover:p	2731	1126			
visibility:m	2731	2729			
wind_speed_10m:ms	2731	366			
y	2731	895			

	first	last	\
ceiling_height_agl:m	NaT	NaT	
clear_sky_energy_1h:J	NaT	NaT	
cloud_base_agl:m	NaT	NaT	
diffuse_rad:W	NaT	NaT	
diffuse_rad_1h:J	NaT	NaT	
direct_rad:W	NaT	NaT	
direct_rad_1h:J	NaT	NaT	
ds	2022-10-28 22:00:00	2023-04-30 19:00:00	
effective_cloud_cover:p	NaT	NaT	
elevation:m	NaT	NaT	
fresh_snow_1h:cm	NaT	NaT	
fresh_snow_3h:cm	NaT	NaT	
fresh_snow_6h:cm	NaT	NaT	

is_day:idx	NaT	NaT
is_in_shadow:idx	NaT	NaT
location	NaT	NaT
precip_type_5min:idx	NaT	NaT
relative_humidity_1000hPa:p	NaT	NaT
sfc_pressure:hPa	NaT	NaT
snow_depth:cm	NaT	NaT
snow_water:kgm2	NaT	NaT
sun_azimuth:d	NaT	NaT
sun_elevation:d	NaT	NaT
super_cooled_liquid_water:kgm2	NaT	NaT
t_1000hPa:K	NaT	NaT
total_cloud_cover:p	NaT	NaT
visibility:m	NaT	NaT
wind_speed_10m:ms	NaT	NaT
y	NaT	NaT

	mean	std	min \
ceiling_height_agl:m	3361.682133	2562.862274	28.0
clear_sky_energy_1h:J	285528.142658	573252.521662	0.0
cloud_base_agl:m	1685.111501	1833.56975	27.5
diffuse_rad:W	26.230813	48.360421	0.0
diffuse_rad_1h:J	94649.301538	172723.786046	0.0
direct_rad:W	32.798068	92.244541	0.0
direct_rad_1h:J	118054.008202	329601.815692	0.0
ds			
effective_cloud_cover:p	66.798654	36.717964	0.0
elevation:m	11.193336	7.806119	6.0
fresh_snow_1h:cm	0.023984	0.147555	0.0
fresh_snow_3h:cm	0.069498	0.352141	0.0
fresh_snow_6h:cm	0.13885	0.57722	0.0
is_day:idx	0.378341	0.472171	0.0
is_in_shadow:idx	0.677133	0.452911	0.0
location			
precip_type_5min:idx	0.076254	0.344931	0.0
relative_humidity_1000hPa:p	71.631472	14.652551	21.7
sfc_pressure:hPa	1009.397775	14.318453	971.15
snow_depth:cm	0.119965	0.56196	0.0
snow_water:kgm2	0.080511	0.19473	0.0
sun_azimuth:d	180.975998	94.222121	14.914
sun_elevation:d	-8.945472	22.095926	-49.887
super_cooled_liquid_water:kgm2	0.035088	0.082444	0.0
t_1000hPa:K	275.52987	4.271781	259.975
total_cloud_cover:p	72.32056	37.085445	0.0
visibility:m	34504.948017	17242.154257	270.3
wind_speed_10m:ms	3.109676	1.782531	0.125
y	179.379421	641.546947	0.0

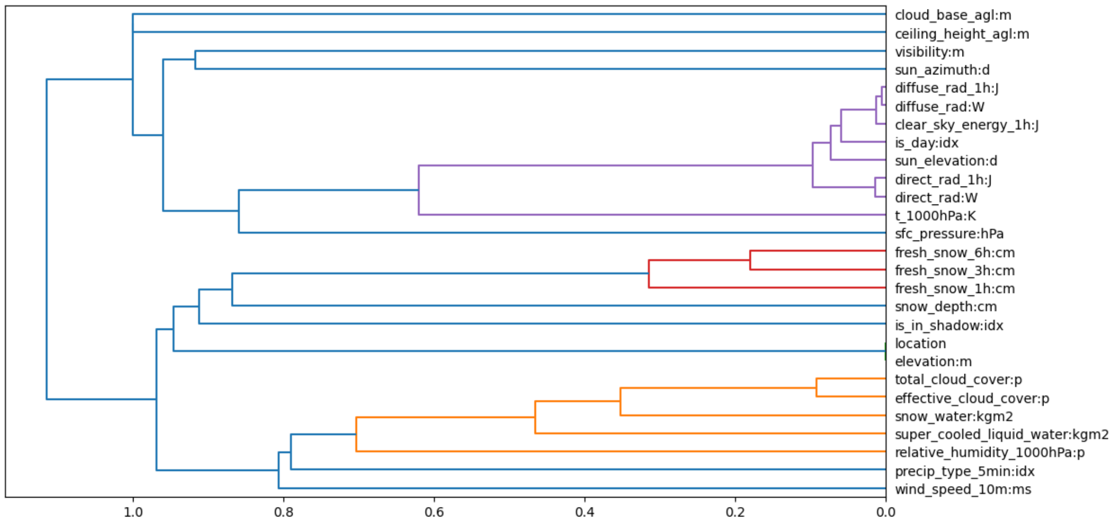
	25%	50%	75%	max	\
ceiling_height_agl:m	1245.55625	2784.275	4919.18125	12294.9	
clear_sky_energy_1h:J	0.0	0.0	220909.2	2551917.2	
cloud_base_agl:m	516.7625	1000.8	2066.9875	10813.7	
diffuse_rad:W	0.0	0.0	32.0	280.5	
diffuse_rad_1h:J	0.0	0.0	116208.0	986147.0	
direct_rad:W	0.0	0.0	4.7625	511.7	
direct_rad_1h:J	0.0	0.0	24326.65	1844204.9	
ds					
effective_cloud_cover:p	36.3125	83.05	99.825	100.0	
elevation:m	6.0	7.0	24.0	24.0	
fresh_snow_1h:cm	0.0	0.0	0.0	2.3	
fresh_snow_3h:cm	0.0	0.0	0.0	4.8	
fresh_snow_6h:cm	0.0	0.0	0.0	6.3	
is_day:idx	0.0	0.0	1.0	1.0	
is_in_shadow:idx	0.0	1.0	1.0	1.0	
location					
precip_type_5min:idx	0.0	0.0	0.0	3.0	
relative_humidity_1000hPa:p	61.775	73.55	82.9625	99.775	
sfc_pressure:hPa	999.1	1009.5	1020.275	1040.6	
snow_depth:cm	0.0	0.0	0.0	4.9	
snow_water:kgm2	0.0	0.0	0.1	2.15	
sun_azimuth:d	101.047372	179.89075	260.65412	347.81226	
sun_elevation:d	-26.72725	-8.178	6.393625	41.09175	
super_cooled_liquid_water:kgm2	0.0	0.0	0.0	0.75	
t_1000hPa:K	272.6625	275.45	278.5	285.825	
total_cloud_cover:p	44.5875	96.775	100.0	100.0	
visibility:m	20902.55	36141.85	48867.949	73937.67	
wind_speed_10m:ms	1.6	2.8	4.2375	9.9	
y	0.0	0.0	40.397706	5043.72	

	dtypes	missing_count	missing_ratio	\
ceiling_height_agl:m	float64	631	0.231051	
clear_sky_energy_1h:J	float64			
cloud_base_agl:m	float64	357	0.130721	
diffuse_rad:W	float64			
diffuse_rad_1h:J	float64			
direct_rad:W	float64			
direct_rad_1h:J	float64			
ds	datetime64[ns]			
effective_cloud_cover:p	float64			
elevation:m	float64			
fresh_snow_1h:cm	float64			
fresh_snow_3h:cm	float64			
fresh_snow_6h:cm	float64			
is_day:idx	float64			
is_in_shadow:idx	float64			
location	object			

```
precip_type_5min:idx                float64
relative_humidity_1000hPa:p         float64
sfc_pressure:hPa                    float64
snow_depth:cm                       float64
snow_water:kgm2                     float64
sun_azimuth:d                       float64
sun_elevation:d                     float64
super_cooled_liquid_water:kgm2      float64
t_1000hPa:K                         float64
total_cloud_cover:p                 float64
visibility:m                        float64
wind_speed_10m:ms                   float64
y                                   float64
```

```
                               raw_type variable_type special_types
ceiling_height_agl:m              float       numeric
clear_sky_energy_1h:J             float       numeric
cloud_base_agl:m                  float       numeric
diffuse_rad:W                     float       numeric
diffuse_rad_1h:J                  float       numeric
direct_rad:W                      float       numeric
direct_rad_1h:J                   float       numeric
ds                             datetime
effective_cloud_cover:p           float       numeric
elevation:m                       float      category
fresh_snow_1h:cm                  float      category
fresh_snow_3h:cm                  float       numeric
fresh_snow_6h:cm                  float       numeric
is_day:idx                        float      category
is_in_shadow:idx                  float      category
location                         object      category
precip_type_5min:idx              float      category
relative_humidity_1000hPa:p       float       numeric
sfc_pressure:hPa                  float       numeric
snow_depth:cm                     float       numeric
snow_water:kgm2                   float       numeric
sun_azimuth:d                     float       numeric
sun_elevation:d                   float       numeric
super_cooled_liquid_water:kgm2    float       numeric
t_1000hPa:K                       float       numeric
total_cloud_cover:p               float       numeric
visibility:m                      float       numeric
wind_speed_10m:ms                 float       numeric
y                                 float       numeric
```

1.0.1 Feature Distance

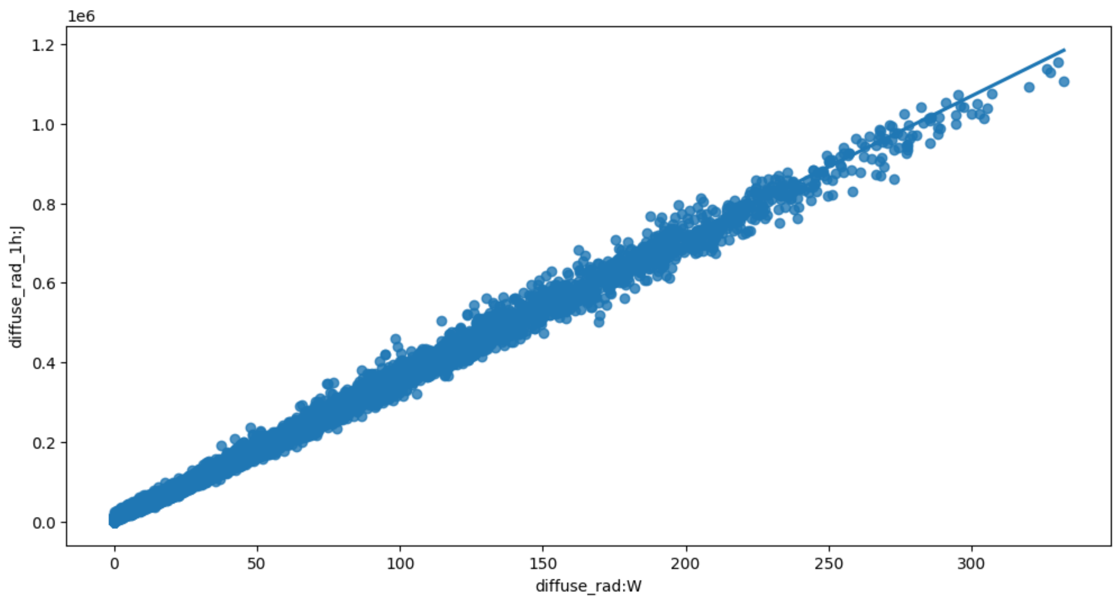


The following feature groups are considered as near-duplicates:

Distance threshold: <= `0.01`. Consider keeping only some of the columns within each group:

- `elevation:m`, `location` - distance `0.00`
- `diffuse_rad:W`, `diffuse_rad_1h:J` - distance `0.00`

Feature interaction between `diffuse_rad:W`/`diffuse_rad_1h:J`

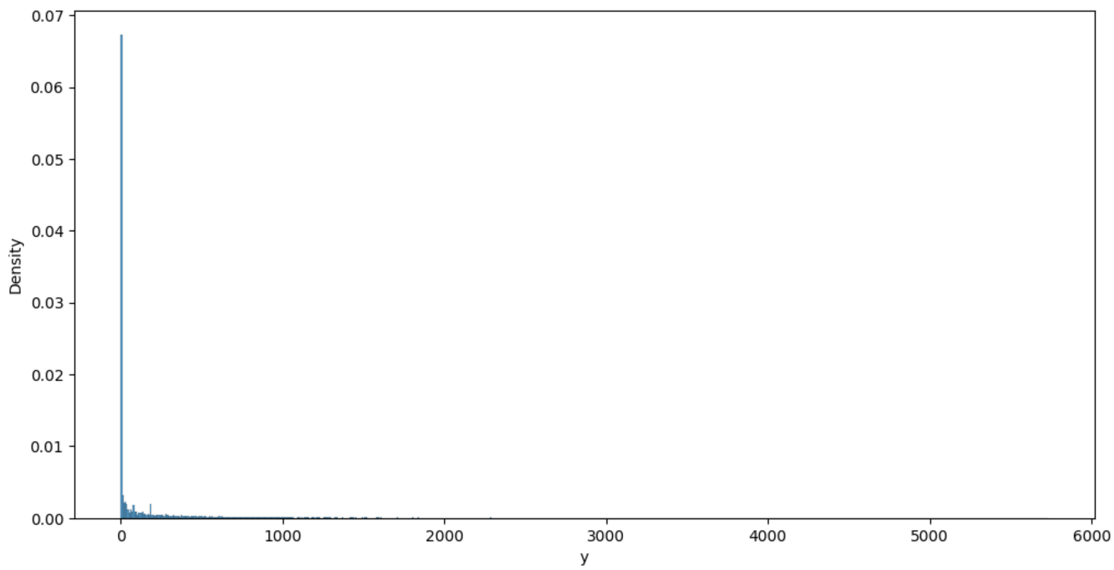


[9]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

1.1 Target variable analysis

```
   count        mean         std  min  25%  50%       75%      max   dtypes  \
y  87486  294.447861  774.531815 -0.0  0.0  0.0  183.7125  5733.42  float64

   unique missing_count missing_ratio raw_type special_types
y   11321                                float
```

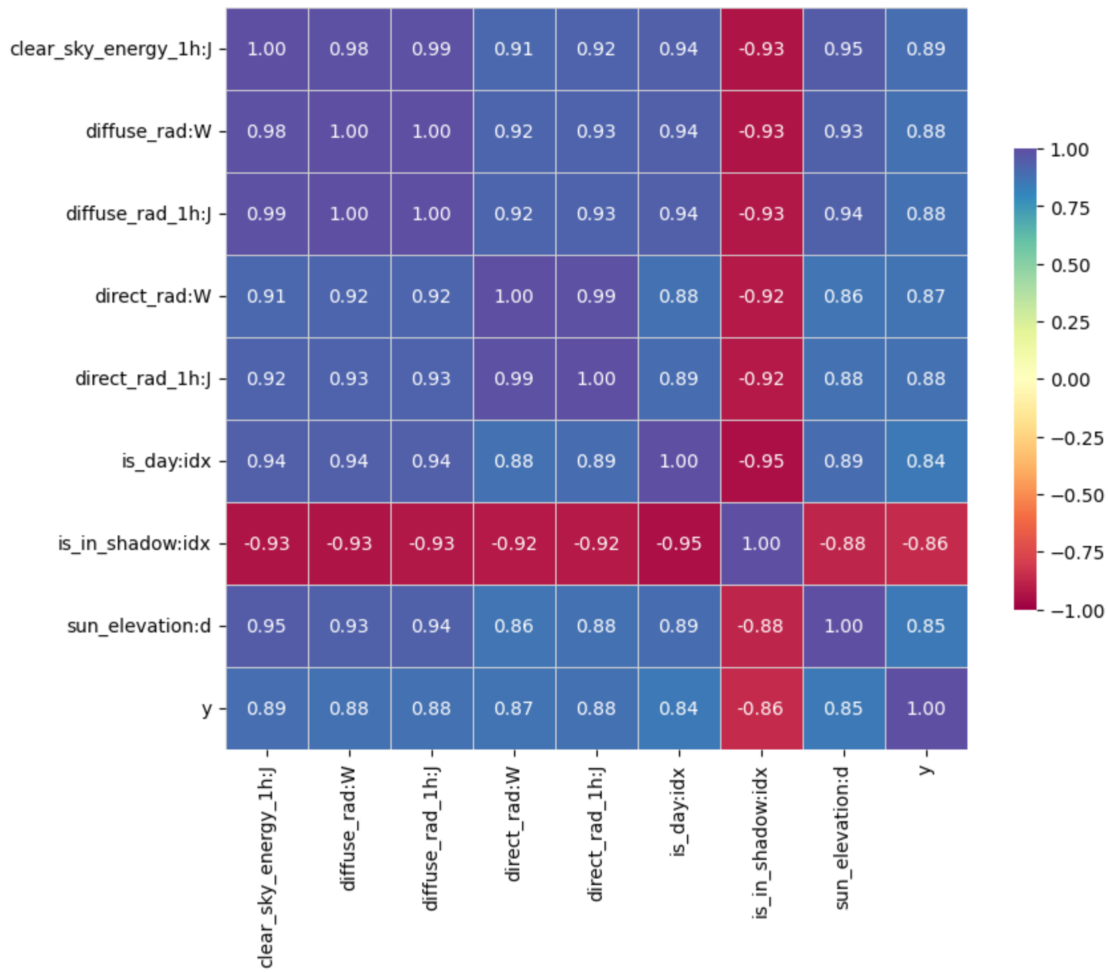


1.1.1 Distribution fits for target variable

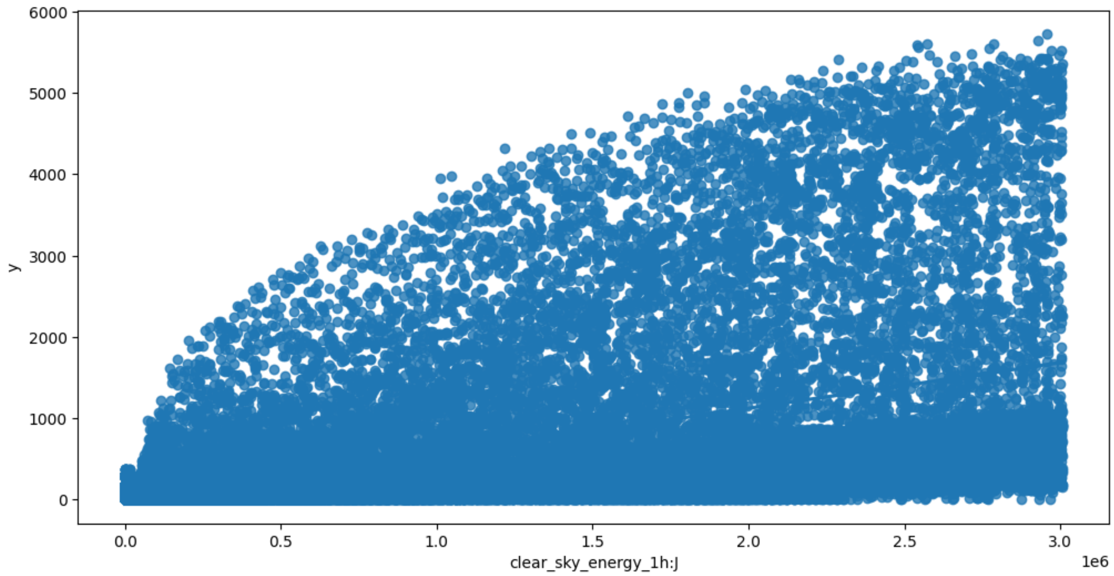
- none of the attempted distribution fits satisfy specified minimum p-value threshold: `0.01`

1.1.2 Target variable correlations

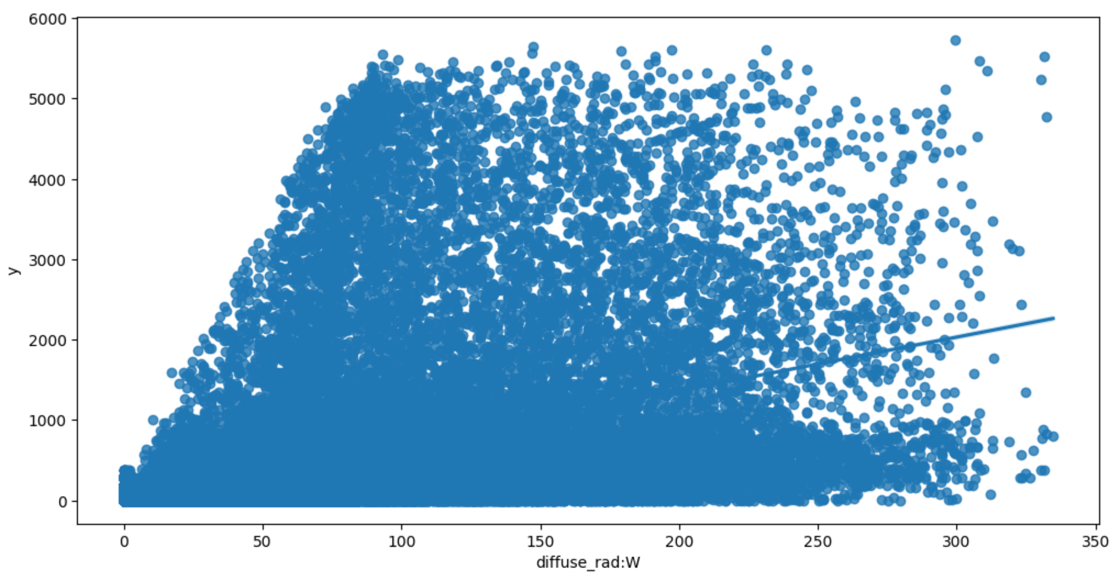
`train_data` - spearman correlation matrix; focus: absolute correlation for `y` >= `0.5`



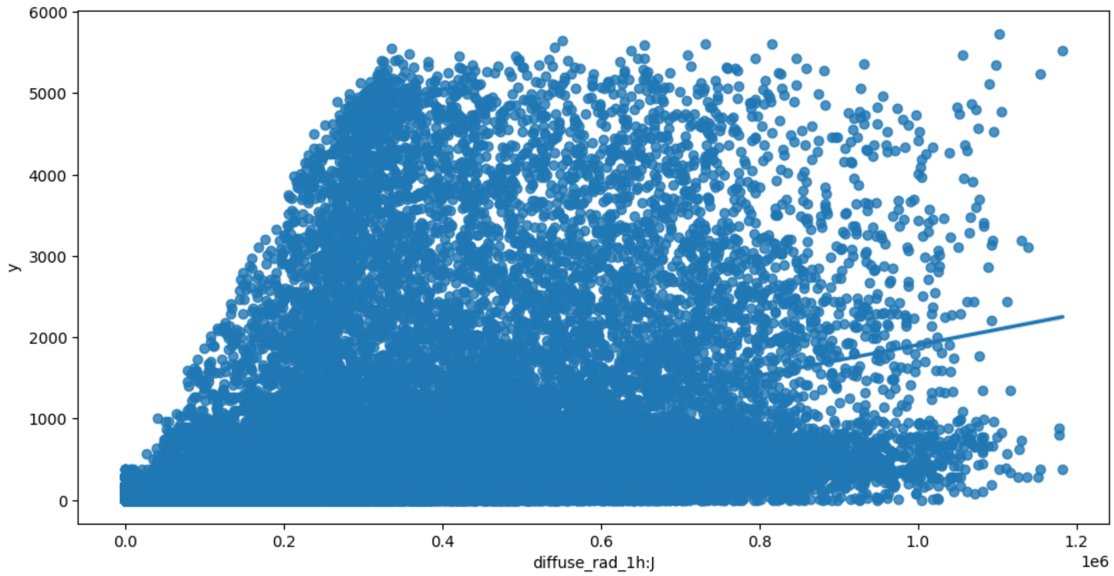
Feature interaction between `clear_sky_energy_1h:J/y` in `train_data`



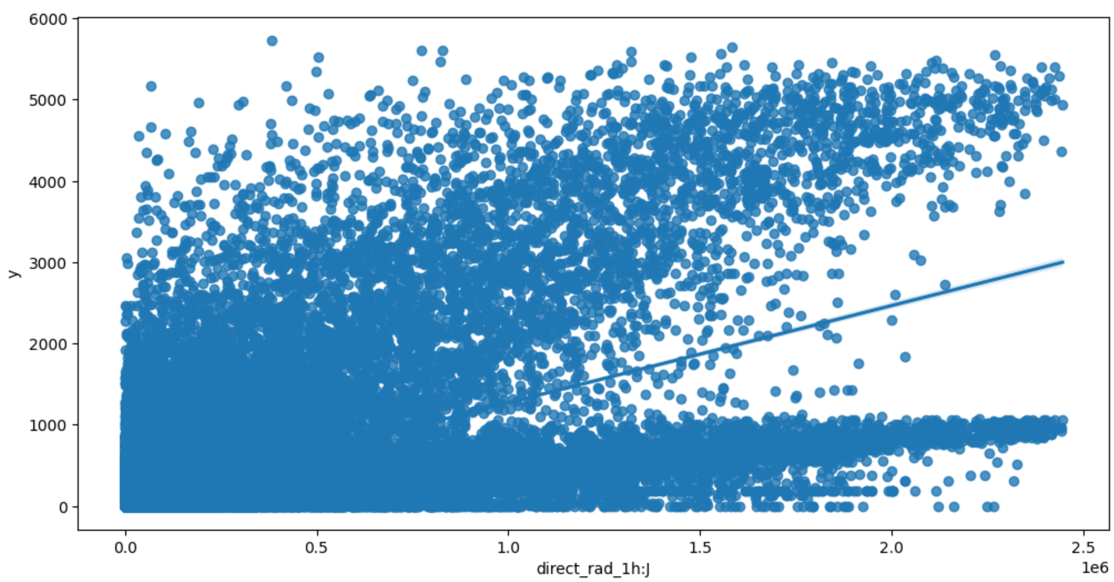
Feature interaction between `diffuse_rad:W/y` in `train_data`



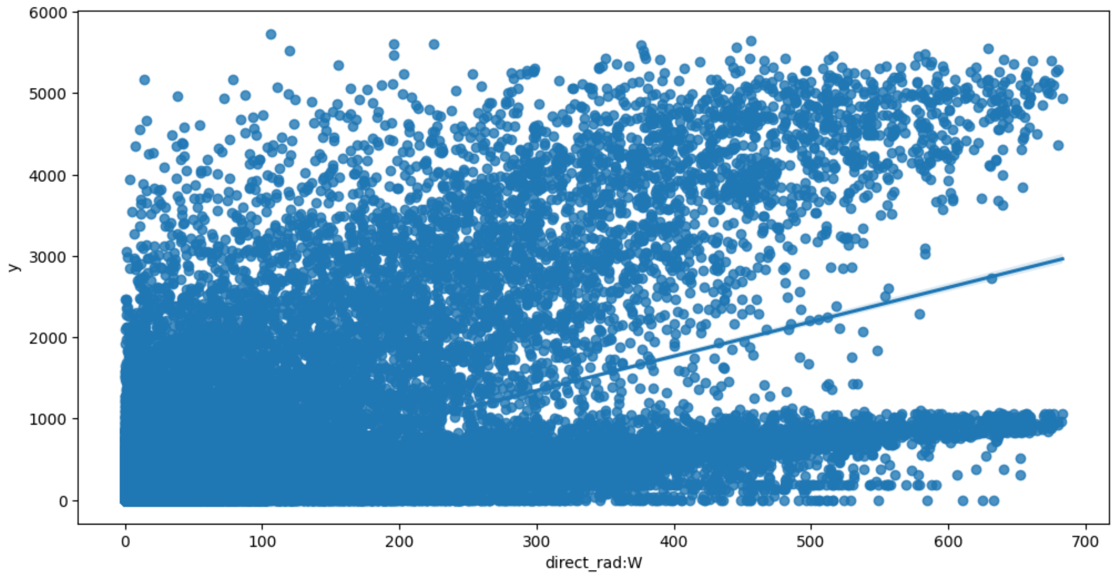
Feature interaction between `diffuse_rad_1h:J/y` in `train_data`



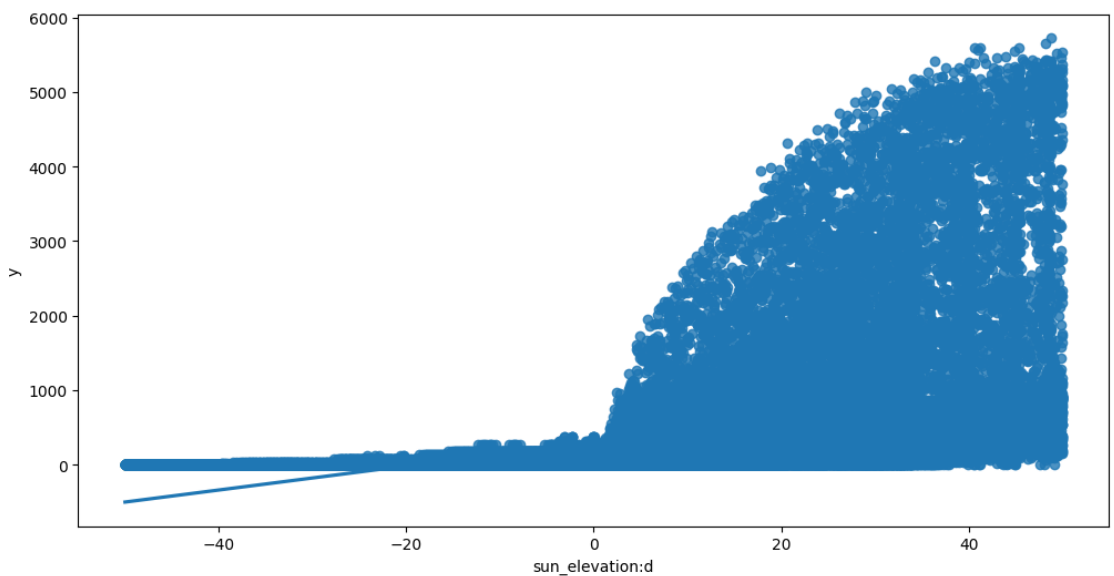
Feature interaction between `direct_rad_1h:J/y` in `train_data`



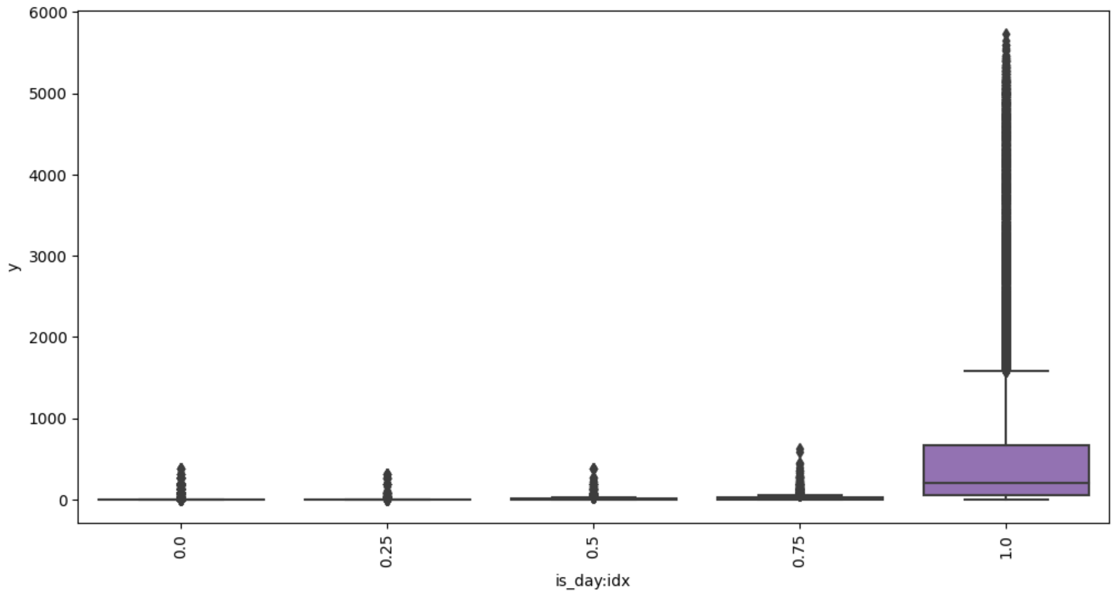
Feature interaction between `direct_rad:W/y` in `train_data`



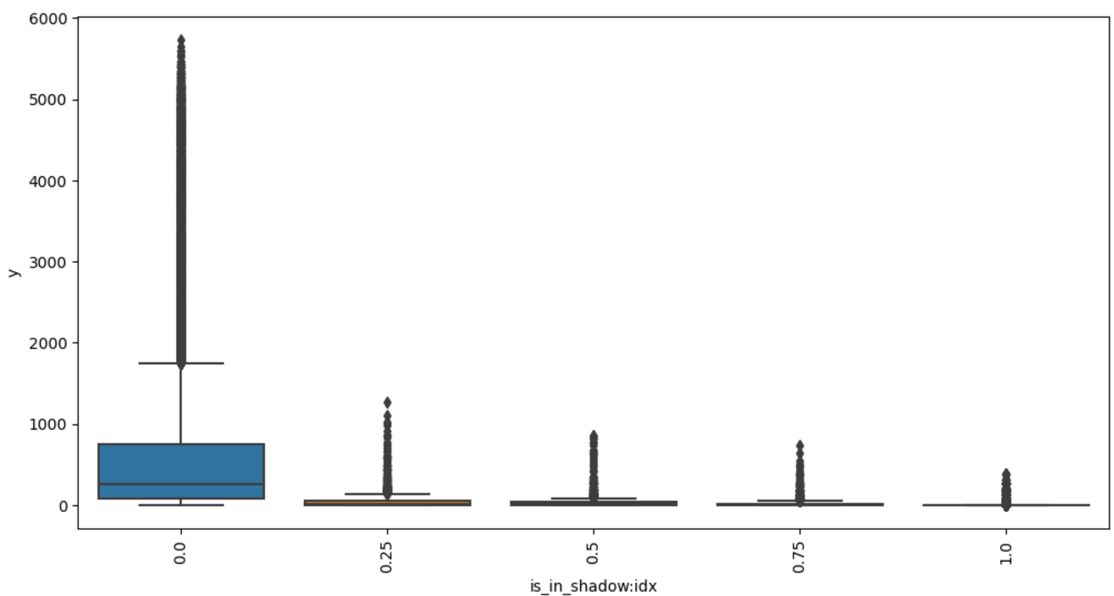
Feature interaction between `sun_elevation:d`/`y` in `train_data`



Feature interaction between `is_day:idx`/`y` in `train_data`



Feature interaction between `is_in_shadow:idx/y` in `train_data`



2 Starting

[10]:
```python
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 99
Now creating submission number: 100
New filename: submission_100
```

[11]:
```python
predictors = [None, None, None]
```

[ ]:
```python
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
```

```
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
        # presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Beginning AutoGluon training … Time limit = 1800s
AutoGluon will save models to "AutogluonModels/submission_100_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   150.33 GB / 315.93 GB (47.6%)
Train Data Rows:    31872
Train Data Columns: 27
Tuning Data Rows:    1093
Tuning Data Columns: 27
Label Column: y
```

```
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 649.68162,
1178.37671)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    131984.69 MB
        Train Data (Original)  Memory Usage: 8.77 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 25 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'cloud_base_agl:m', 'diffuse_rad:W',
'diffuse_rad_1h:J', …]
        Types of features in processed data (raw dtype, special dtypes):
                ('float', []) : 25 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'cloud_base_agl:m', 'diffuse_rad:W',
'diffuse_rad_1h:J', …]
        0.1s = Fit runtime
        25 features in original data used to generate 25 features in processed
data.
        Train Data (Processed) Memory Usage: 6.59 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.14s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
```

```
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}

Training model for location A…

Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 1799.86s of
the 1799.85s of remaining time.
        -140.7607        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 1799.34s of
the 1799.34s of remaining time.
        -140.9568        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1798.88s of the
1798.88s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -100.7752        = Validation score   (-mean_absolute_error)
        32.14s   = Training   runtime
        19.88s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1757.97s of the
1757.97s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -100.7435        = Validation score   (-mean_absolute_error)
```

```
        20.84s   = Training   runtime
        3.9s     = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 1733.76s of
the 1733.76s of remaining time.
        -109.2719        = Validation score   (-mean_absolute_error)
        6.65s    = Training   runtime
        1.15s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1724.67s of the
1724.66s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -106.8026        = Validation score   (-mean_absolute_error)
        187.78s  = Training   runtime
        0.1s     = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 1535.71s of the
1535.71s of remaining time.
        -114.7323        = Validation score   (-mean_absolute_error)
        1.45s    = Training   runtime
        1.15s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 1531.77s of
the 1531.77s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -119.2733        = Validation score   (-mean_absolute_error)
        39.85s   = Training   runtime
        0.48s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 1490.17s of the
1490.17s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -103.765         = Validation score   (-mean_absolute_error)
        5.68s    = Training   runtime
        0.31s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 1482.49s of
the 1482.49s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -99.7353         = Validation score   (-mean_absolute_error)
        129.1s   = Training   runtime
        0.29s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 1351.79s of the
1351.79s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -98.4663         = Validation score   (-mean_absolute_error)
        92.66s   = Training   runtime
        29.64s   = Validation runtime
Repeating k-fold bagging: 2/20
```

```
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 1247.62s of the
1247.62s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -100.3554        = Validation score   (-mean_absolute_error)
        61.35s   = Training   runtime
        40.15s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 1211.12s of the
1211.12s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -99.8771         = Validation score   (-mean_absolute_error)
        46.32s   = Training   runtime
        7.68s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 1181.51s of the
1181.51s of remaining time.
        Fitting 8 child models (S2F1 - S2F8) | Fitting with
ParallelLocalFoldFittingStrategy
```

[ ]:
```python
import matplotlib.pyplot as plt

leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] == loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
 ↪test_data[test_data["location"] == loc]["y"])
        if use_tune_data:
            plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.show()

        return lb
    else:
        return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```

[ ]:
```python
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

[ ]:
```python
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
[ ]: # save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

3 Submit

```
[ ]: import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
[ ]: test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
 ↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

```
[ ]: # predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
 ↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    train_data.loc[train_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
 ↪predictors[i].predict(test_data[test_data["location"] == loc])
```

```
[ ]: # plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
 ↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
 ↪y='prediction', ax=ax, label="past predictions")
    train_data[train_data["location"]==loc].plot(x='ds', y='prediction', ax=ax,␣
 ↪label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction',␣
 ↪ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction',␣
 ↪ax=ax, label="past predictions test")


    # title
    ax.set_title(f"Predictions for location {loc}")
```

```
[ ]: temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())

# Instead of clipping, shift all prediction values up by the largest negative␣
 ↪number.
# This way, the smallest prediction will be 0.
elif shift_predictions:
    for pred in temp_predictions:
```

```
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred["prediction"] = pred["prediction"] - pred["prediction"].min()
        print("Smallest prediction after clipping:", pred["prediction"].min())

elif shift_predictions_by_average_of_negatives_then_clip:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
        # if not nan
        if mean_negative == mean_negative:
            pred["prediction"] = pred["prediction"] - mean_negative

        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

[ ]:
```
# Save the submission DataFrame to submissions folder, create new name based on␣
↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),␣
↪index=False)
print("jall1a")
```

[ ]:
```
train_data_with_dates = TabularDataset('X_train_raw.csv')
train_data_with_dates["ds"] = pd.to_datetime(train_data_with_dates["ds"])
# feature importance
location="A"
split_time = pd.Timestamp("2022-10-28 22:00:00")
estimated = train_data_with_dates[train_data_with_dates["ds"] >= split_time]
estimated = estimated[estimated["location"] == location]
predictors[0].feature_importance(feature_stage="original", data=estimated,␣
↪time_limit=60*10)
```

[ ]:
```
# feature importance
observed = train_data_with_dates[train_data_with_dates["ds"] < split_time]
observed = observed[observed["location"] == location]
```

```
predictors[0].feature_importance(feature_stage="original", data=observed,␣
↪time_limit=60*10)
```

[ ]:
```
# save this running notebook
from IPython.display import display, Javascript
import time

# hei123

display(Javascript("IPython.notebook.save_checkpoint();"))

time.sleep(3)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
↪ipynb"])
```

[ ]:
```
# display(Javascript("IPython.notebook.save_checkpoint();"))
# time.sleep(3)

# subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_with_feature_importance.pdf"),␣
↪"autogluon_each_location.ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])
```

```python
# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
 ↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
 ↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[ ]:

[ ]:

[ ]:

[ ]: